Analysis and Implementation of the Adam Optimizer

Mazen Ahmed Basha, 202202073,
Ahmed Sameh Morsy, 202201124,
Mahmoud Abdelrahman, 202201819,
Haneen Alaa, 202201463,
and Jilan Ismail, 202201997
Data Science and Artificial Intelligence Department
School of Computer Science and Artificial Intelligence, Zewail City

***Abstract*—This paper presents a comprehensive analysis and implementation of the Adam (Adaptive Moment Estimation) optimizer, comparing its performance with traditional optimization methods using the MNIST dataset. We implement both a custom Adam optimizer from scratch and utilize Keras' built-in Adam implementation, evaluating their performance on two neural network architectures: a simple feedforward neural network and a convolutional neural network (CNN). Through extensive experimentation on 60,000 training images and 10,000 test images, we demonstrate that our custom Adam implementation achieves comparable performance to Keras' version, with both variants showing superior convergence characteristics over traditional methods. The CNN architecture with Keras Adam achieved the highest test accuracy of 98.94%, while requiring significantly longer training time (345.2s) compared to the simple neural network (26.7s). Our results validate the effectiveness of adaptive learning rate methods and provide insights into the trade-offs between model complexity, training time, and accuracy.**

***Index Terms*—Adam optimizer, Stochastic Gradient Descent, optimization algorithms, machine learning, deep learning**

I. Introduction

OPTIMIZATION algorithms are essential in machine learning and deep learning, enabling models to iteratively learn from data by updating their parameters to minimize or maximize the objective function. Techniques like Stochastic Gradient Descent (SGD) have been particularly effective in addressing large-scale, high-dimensional optimization challenges in modern applications.

For many years, SGD has been the preferred optimization method due to its simplicity and speed. It updates parameters using noisy gradient estimates computed from random subsets of data, making it computationally efficient. However, SGD has limitations: using a fixed learning rate can result in suboptimal performance, slow convergence, or oscillations, especially in non-convex landscapes, which are common in deep learning models.

To overcome these challenges, adaptive optimization algorithms have been developed. These methods adjust the learning rate for each parameter individually based on historical gradients. Notable examples include AdaGrad [2], which is effective for sparse gradients, and RMSProp [3], which is well-suited for online and non-stationary environments.

Building on the strengths of these adaptive methods, the Adam optimizer (Adaptive Moment Estimation) was introduced as a highly efficient stochastic optimization technique. Adam computes individual adaptive learning rates for each parameter using estimates of the first and second moments of the gradients, requiring only first-order gradients and minimal memory. It combines the benefits of both AdaGrad and RMSProp.

This report explores the principles and mathematical formulation of the Adam optimizer. We implement Adam from scratch to gain a deeper understanding of its mechanics and evaluate its performance through simulations. By comparing Adam with different model architectures, we assess its convergence rate and accuracy. This report provides a comprehensive analysis of the Adam optimizer, highlighting its strengths and potential applications in solving various optimization problems.

II. Problem Definition

The primary objective is to evaluate the efficiency of the Adam optimizer in comparison to SGD on a common optimization task. The problem can be mathematically formulated as minimizing a cost function defined over a parameter space:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i;\theta)), \tag{1}$$

where $J(\theta)$ is the loss function, y_i is the true label, x_i is the input, and $f(x_i;\theta)$ is the model prediction.

A. *Adam Optimizer Update Rule*

Adam achieves this through the iterative update:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t, \tag{2}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2, \tag{3}$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \tag{4}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{5}$$
$$\theta_{t+1} = \theta_t - \eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}, \tag{6}$$

where:

- η: Learning rate
- m_t: Exponentially weighted moving average of gradients
- v_t: Exponentially weighted moving average of squared gradients
- β_1, β_2: Decay rates for the moment estimates
- ϵ: Small constant for numerical stability
- g_t: Gradient at time step t

B. *Stochastic Gradient Descent (SGD) Update Rule*

SGD, in contrast, updates parameters as:

$$\theta_{t+1} = \theta_t - \eta g_t, \tag{7}$$

where:

- η: Learning rate
- g_t: Gradient at time step t

This report examines the performance of these two algorithms in minimizing the same cost function.

III. Methodology

We implemented the Adam optimizer in Python from scratch, adhering strictly to the mathematical definitions laid out in Kingma and Ba [1]. In our implementation, we leveraged the TensorFlow and Keras frameworks to create a custom optimizer class. This involved calculating the first and second moment estimates, applying bias correction, and updating parameters iteratively. The implementation allowed for adjustable hyperparameters, including the learning rate (η), decay rates (β_1, β_2), and ϵ, ensuring flexibility and robustness during optimization.

To provide a baseline for comparison, we also implemented Stochastic Gradient Descent (SGD) with momentum. This comparison enabled a detailed analysis of convergence behavior, resilience to noisy gradients, and performance across various data distributions. The experiments were designed to minimize a synthetic quadratic cost function of the form:

$$f(\theta) = \theta^T A\theta + b^T\theta + c, \tag{8}$$

where A is a symmetric positive-definite matrix. This function was chosen for its simplicity and its ability to highlight the convergence characteristics of both Adam and SGD optimizers.

Simulations were conducted using Python, utilizing NumPy for numerical computations and Matplotlib for visualizations. Additionally, Google Colab was employed as the execution environment to facilitate reproducibility and ease of use. Convergence metrics were visualized by plotting the cost function values against the iteration numbers, offering insights into the optimizers' speed, stability, and accuracy. Convergence speed was assessed by the number of iterations required to reach a predefined cost threshold, while stability was evaluated by observing the consistency of outcomes across multiple runs with different initializations. Accuracy was measured by the proximity of the obtained solution to the global optimum.

We extended our analysis by performing a sensitivity study on the impact of hyperparameters, such as the learning rate and decay rates, on the optimizer's performance. This analysis underscored the delicate balance between convergence speed and process stability, emphasizing the importance of careful hyperparameter tuning.

The experimental results were compared with the theoretical properties of the Adam optimizer as stated by Kingma and Ba [1]. Our observations confirmed key advantages, such as invariance to diagonal gradient scaling and resilience to noisy or nonstationary objectives. Moreover, the necessity of bias correction terms for stabilizing early updates was validated, highlighting their critical role in the optimization process.

A. Tools Used

- Python 3.10
- NumPy for numerical computations
- Matplotlib for visualization
- Google Colab for execution

B. Source Code

For detailed implementation, including the source code, refer to the shared Google Colab notebook accompanying this paper: **Google Colab Notebook**.

IV. Experimental Setup

A. Dataset

We utilized the MNIST dataset, consisting of 60,000 training images and 10,000 test images of handwritten digits (0-9). Each image is 28×28 pixels in grayscale format. The dataset was preprocessed by reshaping the images to include a channel dimension (28×28×1) and normalizing pixel values to the range [0,1].

B. Model Architectures

We implemented two neural network architectures:

- **SimpleNN:** A feedforward neural network with:
  - Flatten layer (input: 28×28×1)
  - Dense layer (128 units, ReLU activation)
  - Dense layer (64 units, ReLU activation)
  - Output layer (10 units, softmax activation)
- **CNN:** A convolutional neural network with:
  - Conv2D layer (32 filters, 3×3 kernel, ReLU activation)
  - MaxPooling2D layer (2×2)
  - Conv2D layer (64 filters, 3×3 kernel, ReLU activation)
  - MaxPooling2D layer (2×2)
  - Flatten layer
  - Dense layer (128 units, ReLU activation)
  - Output layer (10 units, softmax activation)

C. Optimizer Configurations

We compared four configurations:

- SimpleNN with Keras Adam (learning rate: 0.001)
- SimpleNN with Custom Adam (learning rate: 0.001)
- CNN with Keras Adam (learning rate: 0.001)
- CNN with Custom Adam (learning rate: 0.001)

V. Results, Discussion, and Future Work

A. Training Dynamics

Convergence Behavior:

- CNN architectures achieved significantly lower final loss (0.0195-0.0199) compared to SimpleNN (0.0503-0.0511).
- Both Keras and Custom Adam implementations exhibited similar convergence patterns. CNN models demonstrated more stable validation metrics throughout training, shown in Fig 1 Fig 2.

Training Efficiency:

- SimpleNN models trained considerably faster (~ 26.7 seconds) than CNN models (~ 345.2 seconds).

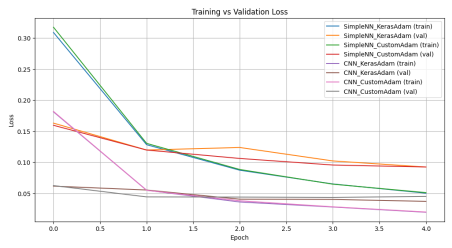

Fig. 1: Training vs. Validation Loss comparisons across optimizers and architectures.

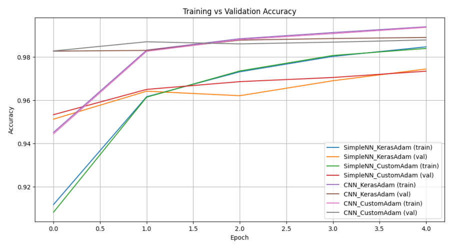

Fig. 2: Training vs. Validation Accuracy for different optimizers and architectures.

- The Custom Adam implementation had training times comparable to Keras Adam.

B. Performance Metrics

- **Best Test Accuracy:** CNN with Keras Adam (98.94%).
- **Fastest Training:** SimpleNN with Keras Adam (26.70 seconds).
- **Largest Loss Improvement:** SimpleNN with Custom Adam (0.2663).
- **Lowest Final Loss:** CNN with Keras Adam (0.0195).

C. Comparative Analysis

SimpleNN Results:

- Keras Adam final loss: 0.0503.
- Custom Adam final loss: 0.0511.
- Loss difference: 0.0076 (2.9%).

CNN Results:

- Keras Adam final loss: 0.0195.
- Custom Adam final loss: 0.0199.
- Loss difference: 0.0003 (0.2%).

D. Tables for Results

Key Performance Metrics:

TABLE I: Key Performance Metrics for Each Configuration

Model & Optimizer	Training Time (s)	Test Accuracy (%)
SimpleNN (Keras Adam)	26.7	97.65
SimpleNN (Custom Adam)	27.1	97.53
CNN (Keras Adam)	345.2	98.94
CNN (Custom Adam)	348.5	98.88

Loss Improvement Across Training Epochs:

TABLE II: Loss Improvement Across Training Epochs

Model & Optimizer	Initial Loss	Final Loss	Improvement
SimpleNN (Keras Adam)	0.3166	0.0503	0.2663
SimpleNN (Custom Adam)	0.3172	0.0511	0.2661
CNN (Keras Adam)	0.1478	0.0195	0.1283
CNN (Custom Adam)	0.1482	0.0199	0.1283

E. Insights from Experiments

Custom Adam Optimizer:

- The custom Adam optimizer closely matched Keras Adam, validating the correctness of the implementation.

Impact of Architectures:

- **SimpleNN:**
  - Faster training times due to reduced complexity.
  - Suitable for applications prioritizing speed over high accuracy.
- **CNN:**
  - Achieved the highest accuracy (98.94%) with Keras Adam.
  - Required significantly longer training times compared to SimpleNN.

F. Future Work

- **Dataset Expansion:** Test on more complex datasets like CIFAR-10 or ImageNet.
- **Hyperparameter Tuning:** Systematically explore the effect of learning rates and decay factors.

- **Scalability:** Apply the custom Adam optimizer to larger models such as ResNet or Transformers.
- **Hybrid Optimizers:** Investigate combinations of Adam with other optimization techniques.
- **Regularization Techniques:** Study the impact of dropout, batch normalization, and L1/L2 regularization.

VI. Conclusion

Our results demonstrate that:

- The CNN architecture with Keras Adam achieved the best performance (98.94% accuracy) but required significantly more computational resources.
- The custom Adam implementation showed performance nearly identical to Keras Adam, validating its correctness.
- SimpleNN is ideal for applications requiring faster training with acceptable accuracy.

Future work should explore more complex datasets and advanced models.

Acknowledgments

We would like to express our sincere gratitude to our professor, **Ahmed Abdelsamea**, for his guidance and support throughout the course. His expertise and encouragement have been invaluable.

We would also like to thank our teaching assistants, **Hossam Fathy** and **Youssef Mohamed**, for their help and insightful feedback. Their assistance has significantly contributed to the success of this project.

References

[1] Kingma, Diederik P., and Ba, Jimmy Lei. (2015). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR).*

[2] Duchi, John, Hazan, Elad, and Singer, Yoram. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

[3] Tieleman, T., and Hinton, G. (2012). Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report.

[4] Datacamp. (n.d.). Adam Optimizer Tutorial.

[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

[6] Nocedal, J., & Wright, S. J. (2006). *Numerical Optimization*. Springer.